# Event-plus-delay Sampling

*Option for IRM KHz digitizer*

Wed, Dec 29, 2004

This note describes a mechanism for limited sampling of KHz digitizer signals for a set of specified delays following a Tevatron clock event.

Recently, a mechanism was added to the IRM system to permit a KHz digitizer hook to be invoked from the KHz digitizer interrupt that occurs following digitization of the last of 64 channels. Use this mechanism to build the results described above. Special code performs the sampling after appropriate delays following a given clock event. An array of sampled readings is accumulated until the last delay is handled.

The following structure includes the required data to specify the clock event and set of delays at which signals are to be sampled. Another structure is used to hold the sampled result data. The structure that guides the local application in capturing the data is as follows:

| *Field* | *Size* | *Meaning* |
|---|---|---|
| `eventNum` | 2 | clock events (arm in hi byte, trigger in lo byte) |
| `nDelays` | 2 | #delays to follow |
| `delay[0]` | 2 | delay in ms units |
| `delay[1]` | 2 | etc. |

The structure of results, for each sampled signal, is as follows:

| *Field* | *Size* | *Meaning* |
|---|---|---|
| `evnt` | 2 | clock events (same as `eventNum` above) |
| `dCnt` | 2 | #delays counter |
| `sample[0]` | 2 | sampled reading |
| `sample[1]` | 2 | etc. |

When reading the sampled data structure, the client can expect to see the `dCnt` field count up to match `nDelays`, when all required samples have been collected. If `dCnt` is less than that, readings are in process of being collected. It is expected that the client will read out the results before another occurrence of the indicated clock event.

***Arm and trigger events***

It is sometimes important to qualify the trigger event by an arm event. The KHz interrupt routine therefore watches for an arm event always, which sets an `armed` flag. If a trigger event occurs when the `armed` flag is set, a trigger condition has occurred that begins the sampling after the specified delays, and the `armed` flag is cleared, in order to prevent subsequent trigger events from being recognized as valid sampling triggers. The input parameter specifying the trigger event as a 16-bit value includes both the arm and trigger event numbers. The hi byte is the arm event; the lo byte is the trigger event. If single event triggering is desired, set both bytes to the same event number, such as `0x2929`. Note that setting the event as `0x0029`, for example, is interpreted as an arm event `0x00` and a trigger event `0x29`.

### *Details*

A local application KHZS sets up the KHz digitizer hook when it initializes itself, usually soon after the front end resets. The client application sets the appropriate specification parameters via the SETDAT protocol. A structure of 64 bytes allows up to 30 delays to be specified. A similar structure for the results allows up to 30 corresponding sampled readings.

The code in the hook routine is given the slot# in the 0–511 range that allows it to find the most recent set of digitized values in the 64 KB circular buffer. It always checks for the occurrence of the trigger condition, which causes it to establish the sampling state and start awaiting the various delays in sequence. As each delay is reached, it copies the latest readings of the selected signals into the result structures. The measurement of elapsed time is done using the microsecond timer counter on the CPU board. Every clock event's occurrence is time stamped in the Event Times table using this same counter, so it is natural to use it for detecting when each delay has been reached. The method of detecting an occurrence of the selected clock event is done by monitoring the Event Times table entry for the selected event. When the time stamp of the selected event changes, the event has just occurred within the last millisecond.

Besides initialization to supply the KHz digitizer hook to the underlying system, the local application has very little to do. But it must watch for possible changes made to the input parameters and update the copy of them used by the KHz routine.

More than one channel can be sampled, in which case more than one result array is built. The specification of which channels are to be sampled can be done via the LA parameters. The current maximum is for 7 channels. If more than this is needed, then additional structures can be defined, using additional instances of the KHZS local application. Note that the scheme as described requires more than one KHz digitizer hook for this case. But multiple hooks are also supported.

As for client access to the structures, one can keep the structures in the static memory block (SM) allocated by the LA. To set up the database entry, one must know the LATBL entry# used by the local application instance, and also the offset into the SM block that applies for the structure sought. The initial structure of parameters is located at 0x40 bytes offset from the SM block, and the result structures are located at offsets 0x100, 0x140, 0x180, etc, for the channels being sampled.

Although access to the result structure can be shared among multiple users without interference, it is likely that only one user will be interested, as only that user specified the clock event and array of delays of interest.

### *More details*

The KHZS local application was written to operate as described above, using the new KHz digitizer hook facility supported by the IRM system code. The client access to the input parameters has been defined as three Acnet devices. Two are used for the arm and trigger clock events and the number of delays. The third is used for the array of sample delays, in millisecond units internally. The reading of the number of delays is actually taken from the result buffer, so it can be monitored as it counts from 0 to the current `nDelays` setting. When it reaches that value, the client should read the result samples before another occurrence of the trigger event clears the count of delays.

In order to provide for nonvolatile storage of the input parameters, so that a reboot of the front end does not cause them to be lost, a copy of the input parameters is kept in a sequential series of analog channels. When the node is booted, these channel readings are used to establish the initial values of the working input parameters.

To make this work, setting the input parameters targets a buffer that the LA monitors for changes. If any change is detected, the series of analog channel readings is loaded with the new values. After that point in the 15 Hz logic of the LA, the series of analog channel readings is matched against the active set of parameters, and if any change is noted, the new set of parameters is carefully updated for the KHz routine to use.

There is another Acnet device for each of the result arrays of sampled channel readings. The maximum number of delays, also the maximum number of result samples, is 30.

Timing measurements were done for the KHz activity related to this LA. When an event is noted, the time required to process it is about 10 $\mu$s. When the current delay has been reached, the time to sample two signals is less than 20 $\mu$s. These are small execution times compared to the 1000 $\mu$s between executions of the KHz routine.

The array of delays is assumed in order, shortest to longest; if they are not, incorrect samples may result. If two consecutive delays are equal, both samples are identical.

The analog channel numbers for the 64-channel KHz digitizer used in IRMs range from `0x0100–0x013F`. If a channel is specified in the LA parameters that is not in this range, the current reading in the 15 Hz data pool is sampled. If the channel number is zero, it is ignored and not sampled.